DAM
Desarrollo de Aplicaciones Multiplataforma
2º Curso

# AD
# Acceso a Datos

UD 8
Programación de componentes
de acceso a datos
Parte 2

IES BALMIS
Dpto Informática
Curso 2019-2020
Versión 1 (12/2019)

# UD8 – Programación de componentes de acceso a datos

## ÍNDICE

# 7. Programación de aplicaciones en Servidores Web

## 7.1 Lenguaje de script de Servidor

Cuando creamos aplicaciones web, necesitamos un servidor que además de ofrecer archivos (html, css, js, …) ejecute código que aporte funcionalidad a la aplicación.

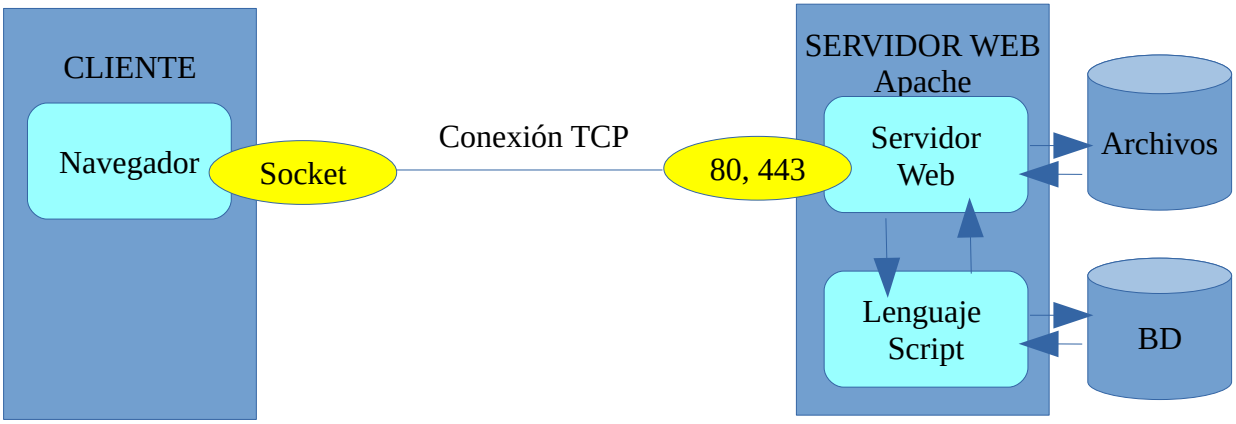
**Lenguaje del lado del servidor** es aquel que se ejecuta en el servidor web, inmediatamente antes de que el sitio web se envíe a través de Internet al usuario.

Los sitios web que se ejecutan en el servidor pueden realizar un amplio abanico de tareas hasta formar el propio sitio web que va a ver el usuario: acceso a base de datos, conexión en red, …

Los lenguajes de lado servidor más ampliamente utilizados para el desarrollo de páginas dinámicas son el PHP, ASP y JSP.

## 7.2 Apache y lenguaje de script PHP

Para poder implantar una solución Web que utilice en el servidor lenguaje PHP, necesitaremos un servidor web como **Apache** que ejecute el código y devuelva el resultado al navegador cliente:




***Lenguaje de Script PHP***

Aunque existen varios, en nuestro caso utilizaremos Apache +PHP.

Podemos instalar manualmente Apache y PHP de forma independiente y luego configurar Apache para utilice PHP. Otra opción para desarrollo es utilizar aplicaciones con Apache y PHP preconfigurados como XAMPP.

## *HTML*

Veamos un ejemplo de página web básica en HTML con una tabla de datos:

**ejemplo_table.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8"/>
  <title>DAM - Tabla de Datos</title>
</head>
<body>
	<div>
		<p>Número de registros: 7</p>
	</div>
	<table border="1" style="border-collapse:collapse">
	  <thead style="background-color:orange">
		<tr>
			<td>ID LIBRO</td>
			<td>TÍTULO</td>
			<td>AUTOR</td>
		</tr>
	 </thead>
	  <tbody>
		<tr>
			<td>1</td>
			<td>Macbeth</td>
			<td>William Shakespeare</td>
		</tr>
		<tr>
			<td>2</td>
			<td>La Celestina (Tragicomedia de Calisto y Melibea)</td>
			<td>Fernando de Rojas</td>
		</tr>
		<tr>
			<td>3</td>
			<td>El Lazarillo de Tormes</td>
			<td>Anónimo</td>
		</tr>
		<tr>
			<td>4</td>
			<td>20.000 Leguas de Viaje Submarino</td>
			<td>Julio Verne</td>
		</tr>
		<tr>
			<td>5</td>
			<td>Alicia en el País de las Maravillas</td>
			<td>Lewis Carrol</td>
		</tr>
		<tr>
			<td>6</td>
			<td>Cien Años de Soledad</td>
			<td>Gabriel García Márquez</td>
		</tr>
		<tr>
			<td>7</td>
			<td>La tempestad</td>
			<td>William Shakespeare</td>
		</tr>
	  </tbody>
	</table>
</body>
</html>
```

## ***PHP***

Para el caso anterior, una opción es disponer de los datos en un array y recorrerlo par mostrar la tabla:

**ejemplo_table.php**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8"/>
  <title>DAM - Tabla de Datos</title>
</head>
<body>
<?php

      $datos=array();

      $registro=array();
      $registro['id']=1;
      $registro['titulo']='Macbeth';
      $registro['autor']='William Shakespeare';
      $datos[]=$registro;

      $registro=array();
      $registro['id']=2;
      $registro['titulo']='La Celestina (Tragicomedia de Calisto y Melibea)';
      $registro['autor']='Fernando de Rojas';
      $datos[]=$registro;

      $registro=array();
      $registro['id']=3;
      $registro['titulo']='El Lazarillo de Tormes';
      $registro['autor']='Anónimo';
      $datos[]=$registro;

      $registro=array();
      $registro['id']=4;
      $registro['titulo']='20.000 Leguas de Viaje Submarino';
      $registro['autor']='Julio Verne';
      $datos[]=$registro;

      $registro=array();
      $registro['id']=5;
      $registro['titulo']='Alicia en el País de las Maravillas';
      $registro['autor']='Lewis Carrol';
      $datos[]=$registro;

      $registro=array();
      $registro['id']=6;
      $registro['titulo']='Cien Años de Soledad';
      $registro['autor']='Gabriel García Márquez';
      $datos[]=$registro;

      $registro=array();
      $registro['id']=7;
      $registro['titulo']='La tempestad';
      $registro['autor']='William Shakespeare';
      $datos[]=$registro;

?>
```

A continuación tendríamos el bucle para mostrar en HTML las líneas de la tabla:

```
      <div>
            <p>Número de registros: <?php echo count($datos); ?></p>
      </div>

<?php if (count($datos)>0) { ?>
      <table border="1" style="border-collapse:collapse">
        <thead style="background-color:orange">
            <tr>
                  <td>ID LIBRO</td>
                  <td>TÍTULO</td>
                  <td>AUTOR</td>
            </tr>
        </thead>
        <tbody>
<?php
        for ($i=0; $i<count($datos); $i++) {
             $registro=$datos[$i];
?>
          <tr>
            <td><?php echo $registro['id']; ?></td>
            <td><?php echo $registro['titulo']; ?></td>
            <td><?php echo $registro['autor']; ?></td>
          </tr>
<?php
        }
?>
        </tbody>
      </table>
<?php
      } else {
?>
      <div>
            <p>No hay datos que mostrar</p>
      </div>

<?php
      }
?>

</body>
</html>
```

Como podemos observar el código de script PHP está intercalado con el lenguaje de marcas HTML, por lo que al final, obtendremos un HTML resultante de la ejecución, manteniendo el HTML existente.

## ***PHP con acceso a BD***

La gran potencia de los lenguajes script en servidores web radica en la utilización de bases de datos, es decir, para cargar los datos utilizaremos código que leerá a través de SQL de una BD.

Cambiaremos por tanto solo la primera parte:

**ejemplo_select.php**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8"/>
  <title>DAM - Tabla de Datos</title>
</head>
<body>
<?php

      $datos=array();

            $servername = "localhost";
            $username = "root";
            $password = "1234";
            $dbname = "bibliotecah";

            // Create connection
            $conn = new mysqli($servername, $username, $password, $dbname);
            // Check connection
            if ($conn->connect_error) {
                  die("Error en la conexión: " . $conn->connect_error);
            }

            $sql = "SELECT * FROM libros";
            $result = $conn->query($sql);

            if ($result->num_rows > 0) {
                  while($registro = $result->fetch_array()) {
                        $campos=array();
                        foreach ($registro as $campo => $valor) {
                              if (!is_int($campo)) {
                                    $campos[$campo]=$valor;
                              }
                        }
                        $datos[]=$campos;
                  }
            }
            $conn->close();


?>
```

El resultado es el mismo, pero ahora cambiando el contenido de la base de datos, el resultado de la página cambiará.

También podemos crear el contenido de los datos en formato JSON, en vez de HTML.

PHP proporciona funciones para crear este contenido fácilmente:

**ejemplo_select_json.php**

```php
<?php

	$datos=array();

		$servername = "localhost";
		$username = "root";
		$password = "1234";
		$dbname = "bibliotecah";

		// Create connection
		$conn = new mysqli($servername, $username, $password, $dbname);
		// Check connection
		if ($conn->connect_error) {
			die("Error en la conexión: " . $conn->connect_error);
		}

		$sql = "SELECT * FROM libros";
		$result = $conn->query($sql);

		if ($result->num_rows > 0) {
			while($registro = $result->fetch_array()) {
				$campos=array();
				foreach ($registro as $campo => $valor) {
					if (!is_int($campo)) {
						$campos[$campo]=$valor;
					}
				}
				$datos[]=$campos;
			}
		}
		$conn->close();

    header('Content-Type: application/JSON');
    echo json_encode($datos, JSON_PRETTY_PRINT);
?>
```

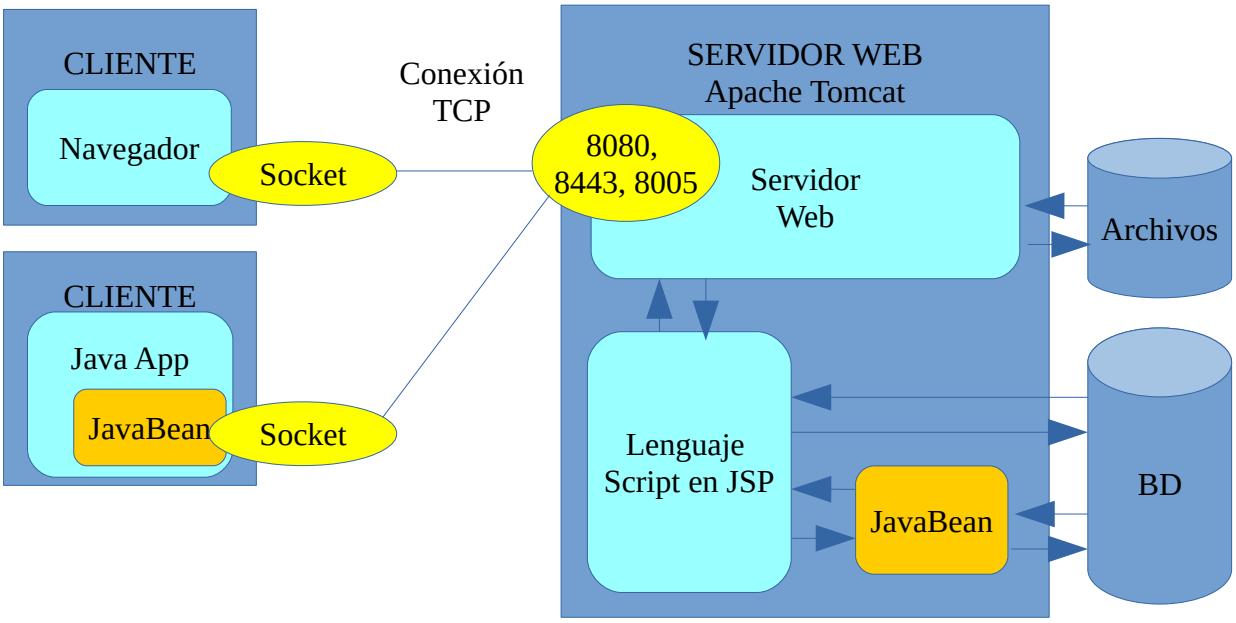
## 7.3 Apache Tomcat

**JSP (JavaServer Pages)** es una tecnología que permite generar documentos en formato HTML de manera dinámica, utilizando el lenguaje de programación Java.

Para poder utilizar JSP (Java Server Pages) vamos a utilizar Apache Tomcat, uno de los servidores más conocidos por los desarrolladores de Java.

Existen también versiones preconfiguradas como la que incorpora XAMPP pero nosotros utilizaremos la oficial para disponer de la última versión.

**Apache Tomcat** es un servidor web que utiliza como lenguaje script del servidor JSP (Java Server Pages).




Podemos observar en el esquema que Apache Tomcat con JSP incorpora la posibilidad de acceder a JavaBean.

**JavaBean** es un componente de software para la plataforma Java SE.

Especificaciones que se debe tener en cuenta para que una clase sea un JavaBean:

- Debe tener un **constructor vacío**
- Debe implementar la interfaz **Serializable**
- Las **propiedades/atributos** deben ser **privados**.
- Debe tener métodos **getters o setters** o ambos que permitan acceder a sus propiedades

Esto nos permite utilizar archivos (JavaBean) empaquetados en JAR para ampliar las posibilidades de la aplicación, como muchas de las librerías qaue hemos añadido a nuestros proyectos.

Para instalar Apache Tomcat, basta con descargar el software y descomprimirlo en una carpeta del servidor.

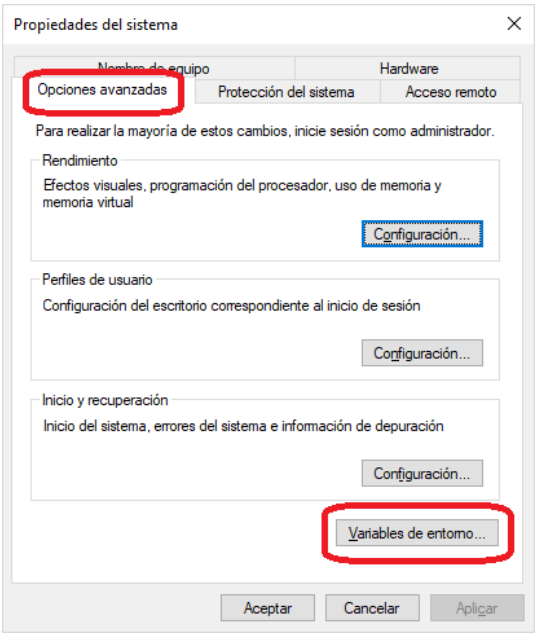
| ***Apache Tomcat – Web Oficial*** |
| --- |
| https://tomcat.apache.org/ |

## ***Configuración Apache Tomcat - Variables***

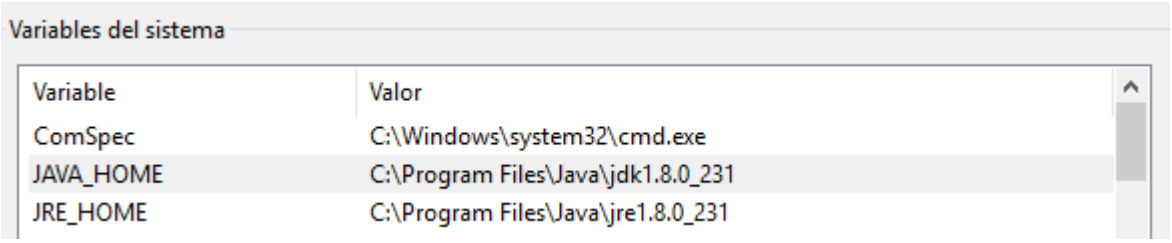
Es necesario añadir dos variables al sistema para que pueda localizar Java.

```
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_XXX
JRE_HOME=C:\Program Files\Java\jre1.8.0_XXX
```

Para ello abriremos las propiedades del equipo con el comando **sysdm.cpl**. En "Opciones avanzadas", pulsar en "Variables de Entorno"




Luego añadiremos las dos variables en el apartado de "Variables de sistema"




Estas variables hay que actualizarlas si se actualiza la versión de Java.

## *Configuración Apache Tomcat – Carpetas*

Apache tiene varias carpetas importantes:

| | |
|---|---|
| **tomcat\webapps\ROOT** | Alojamiento de las páginas HTML y JSP |
| **tomcat\conf** | Archivos de configuración del servidor |
| **tomcat\lib** | Archivos jar cargados en memoria por el servidor y disponibles para los archivos JSP.<br>Cada vez que se incluya uno nuevo, es necesario reiniciar el servidor.<br>Los drivers JDBC irán en este directorio |

## *Configuración Apache Tomcat – Puertos*

Los puertos que usa por defecto son: 8080, 8005 y 8443.

Estos puertos pueden cambiarse en el archivo:

**`tomcat\conf\server.xml`**

Recuerda que el comando para ver los puertos ocupados en windows es:

```
C:\> netstat -a -p TCP | find "LISTENING"
```

## *Configuración Apache Tomcat – Usuarios*

Para configurar Apache Tomcat debemos crear un usuario administrador. Para ello

Para ello editaremos y añadiremos al archivo:

**`tomcat\conf\tomcat-users.xml`**

```
<tomcat-users ...>
  ...
  <user roles="manager-gui, manager-script,admin-gui, admin-script"
        username="tomcat"
        password="1234" />
  ...
</tomcat-users>
```

## ***Configuración Apache Tomcat – Arrancar y parar***

Para arrancar el servicio tenemos el archivo bat:

**`tomcat\bin\startup.bat`**

Para parar el servicio tenemos el archivo bat:

**`tomcat\bin\shutdown.bat`**

## ***Configuración Apache Tomcat – Añadir componentes JAR***

Todos los archivos JAR que vayan a utilizar nuestra páginas web en JSP hay que copiarlos a la carpeta:

**`tomcat\lib`**

Y luego reiniciar el servicio

## ***Configuración Apache Tomcat – URL***

Tendremos varias URL predefinidas para acceder a Apache Tomcat:

`http://localhost:8080/`

URL Pública que navega sobre el contenido de la carpeta tomcat\webapps\ROOT

`http://localhost:8080/manager/html`

URL Administración que navega sobre el Manager GUI

## ***Configuración Apache Tomcat – Prueba***

Crearemos la carpeta donde alojaremos nuestros archivos HTML, CSS, JS y JSP.

**`tomcat\webapps\ROOT\ad`**

| **Prueba del servidor Apache Tomcat** |
| --- |
| Arranca el servicio de Apache Tomcat.<br><br>Prueba a copiar una página html en la carpeta "**tomcat\webapps\ROOT\ad**" y acceder a ella desde el navegador. |

## 7.4 Lenguaje de script JSP

Haciendo la equivalencia con PHP, podríamos tener la tabla de libros del ejemplo:

**ejemplo_table.jsp**

```
<%@page import="java.util.regex.Pattern"%>
<%@page import="java.util.ArrayList"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>DAM - Tabla de Datos</title>
</head>
<body>
<%
  ArrayList<String> datos = new ArrayList<>();

  datos.add("1|Macbeth|William Shakespeare");
  datos.add("2|La Celestina (Tragicomedia de Calisto y Melibea)|Fernando de Rojas");
  datos.add("3|El Lazarillo de Tormes|Anónimo");
  datos.add("4|20.000 Leguas de Viaje Submarino|Julio Verne");
  datos.add("5|Alicia en el País de las Maravillas|Lewis Carrol");
  datos.add("6|Cien Años de Soledad|Gabriel García Márquez");
  datos.add("7|La tempestad|William Shakespeare");
%>
       <div>
            <p>Número de registros: <%= datos.size() %></p>
       </div>

<%     if (datos.size()>0) { %>
       <table border="1" style="border-collapse:collapse">
         <thead style="background-color:orange">
            <tr>
                  <td>ID LIBRO</td>
                  <td>TÍTULO</td>
                  <td>AUTOR</td>
            </tr>
      </thead>
         <tbody>
<%
            for (int i=0; i<datos.size(); i++) {
                  String[] registro = datos.get(i).split(Pattern.quote("|"));
%>
                        <tr>
                              <td><% out.println(registro[0]); %></td>
                              <td><% out.println(registro[1]); %></td>
                              <td><% out.println(registro[2]); %></td>
                        </tr>
<%
            }
%>
         </tbody>
       </table>
<%
            } else {
%>
       <div>
            <p>No hay datos que mostrar</p>
       </div>

<%
        }
%>
</body>
</html>
```

## *JSP con acceso a BD*

Cambiaremos ahora solo la primera parte para leer de la base de datos biblioteca como ya lo habíamos realizado en temas anteriores:

**ejemplo_select.jsp**

```
<%@page import="java.sql.Connection"%>
<%@page import="java.util.regex.Pattern"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.sql.Statement"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.DriverManager"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>DAM - Tabla de Datos</title>
</head>
<body>
<%
    //Estructura de datos para almacenar
    ArrayList<String> datos = new ArrayList<>();

    // Conexión a la BD
    String url;

    Class.forName("com.mysql.jdbc.Driver");
    url = "jdbc:mysql://localhost:3306/bibliotecah?autoReconnect=true";
    url += "&useSSL=false&zeroDateTimeBehavior=convertToNull";
    String usuario = "root";
    String password = "1234";
    Connection con = DriverManager.getConnection(url, usuario, password);

    // Crear Statement de la Consulta
    String sentenciaSQL = "SELECT id, titulo, autor FROM libros";
    Statement statement = con.createStatement();

    // Resulset
    ResultSet rs = statement.executeQuery(sentenciaSQL);
    while (rs.next()) {
        datos.add(String.valueOf(rs.getInt(1))+"|"+rs.getString(2)+"|"+rs.getString(3));
    }
    rs.close();

    // Cerrar conexión
    con.close();
%>

...
```

Para probar estos dos ejemplos, copiaremos los archivos a la carpeta:

**tomcat\webapps\ROOT\ad**

Recuerda que para que funcione correctamente el **ejemplo_select.jsp** necesita de la librería jar del driver JDBC de mysql, por lo que previamente habrá que copiarla a la carpeta de librerías de tomcat y posteriormente reiniciar el servicio:

**tomcat\lib\mysql-connector-java-5.1.48-bin**

# 8. Creación de aplicaciones web

## 8.1 NetBeans con Apache Tomcat

Vamos a realizar nuestro primer proyecto Web usando Apache Tomcat.

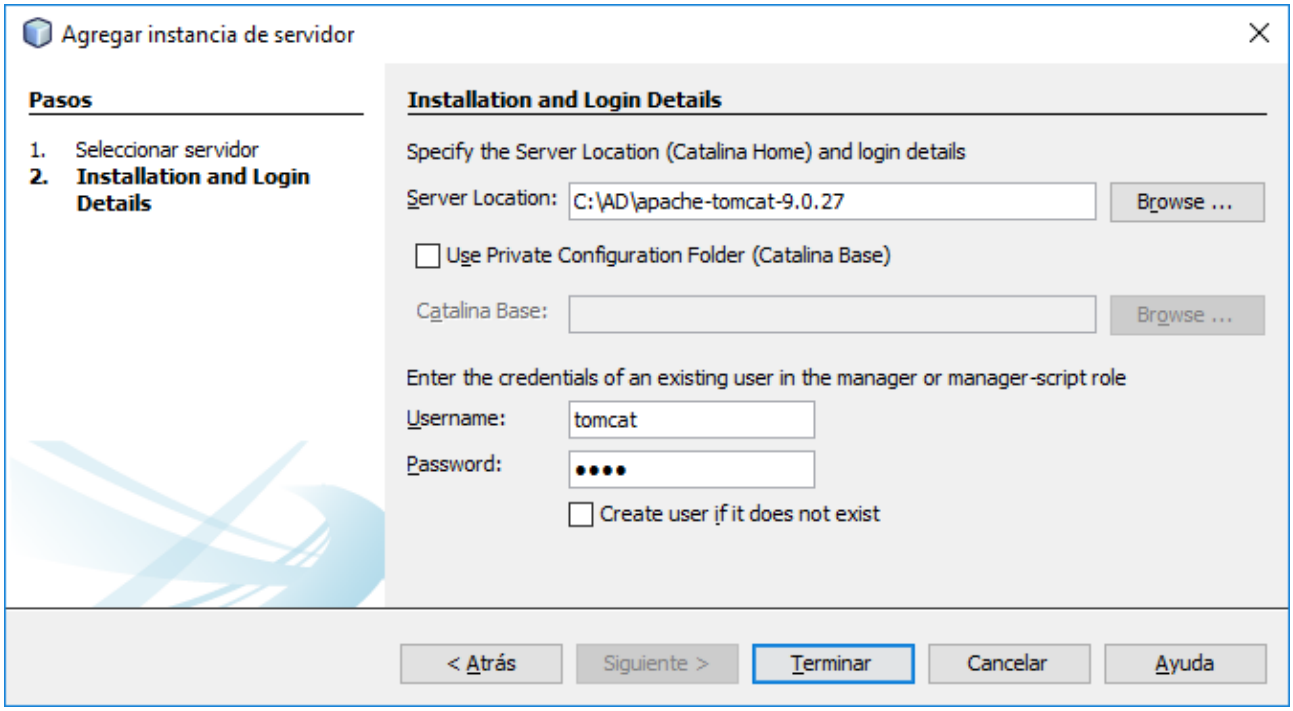
En desarrollo, es recomendable tener el servicio parado y que NetBeans gestione el arranque y la parada de nuestro servidores.

***Añadir el servidor Apache Tomcat a Netbeans***

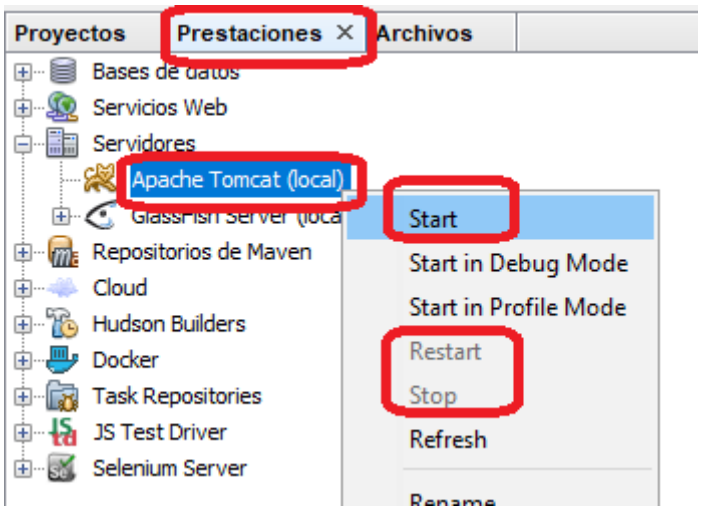
En la pestaña de Prestaciones añadiremos nuestro servidor de

**Apache Tomcat (local)**

indicando el usuario administrador **tomcat** con su contraseña **1234** que hemos configurado anteriormente.



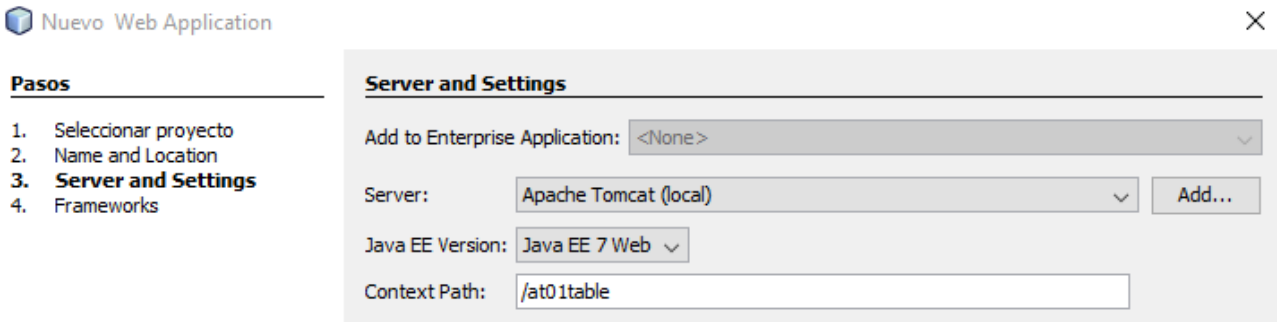

Desde NetBeans podemos iniciar, resiniar o parar el servidor utilizando el botón derecho.



## ***Crear proyecto de Java Web => Web Application***

Vamos a crear el proyecto **AT01Table** que ejecute el código de **ejemplo_table.jsp**.

Crearemos un proyecto de tipo "Java Web => Web Application" poniendo como path raíz del App Web (Context Path Root) **at01table** y como servidor nuestro **Apache Tomcat (local).**




Ahora solo queda copiar el contenido de nuestro **ejemplo_table.jsp** en el archivo **index.jsp** que ha generado NetBeans.

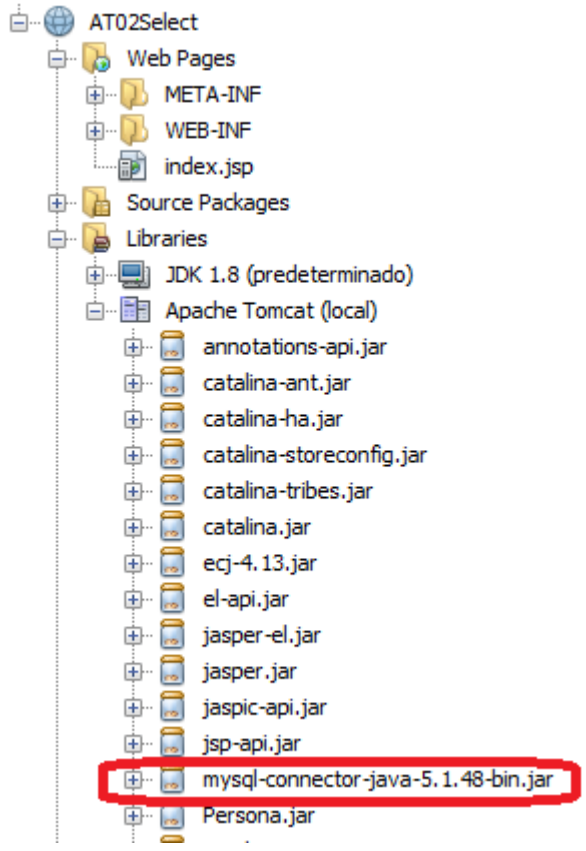
Cuando ejecutemos el proyecto, si Apache Tomcat está parado, se iniciará (Start). Hay que tener en cuenta que al cerrar NetBeans se parará (Stop).

## ***Crear proyecto de Java Web => Web Application***

Vamos a crear el proyecto **AT02Select** que ejecute el código de **ejemplo_select.jsp**.

Realizaremos el mismo proceso que en el anterior proyecto, pero poniendo como path raíz del App Web (Context Path Root) **at02select** y copiando el contenido de nuestro **ejemplo_select.jsp** en el archivo **index.jsp.**

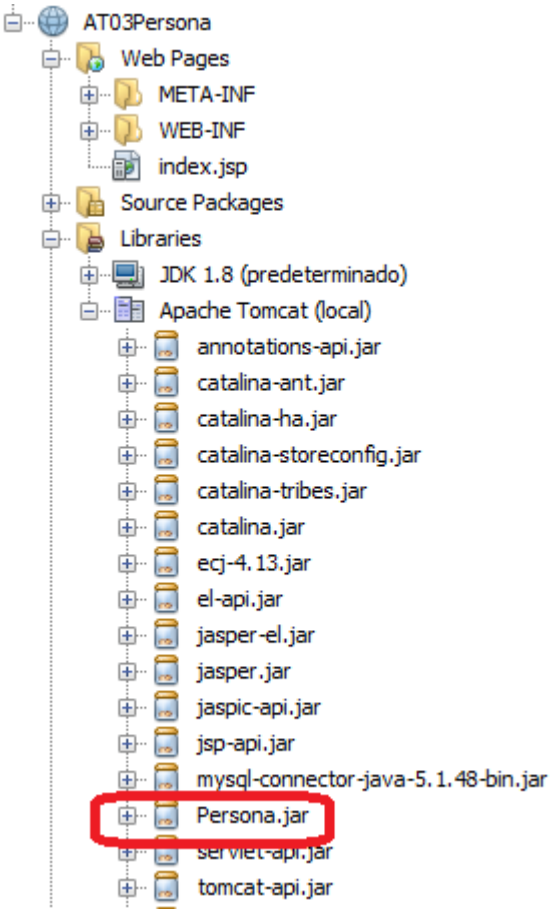
No es necesario copiar la librería jar del driver de mysql a la aplicación porque ya está en **tomcat\lib** del Apache Tomcat y si desplegamos las librerías incluidas en el servidor la podremos ver.
*(ver imagen lateral)*



## ***Crear proyecto de Java Web => Web Application***

Vamos a crear el proyecto **AT03Persona** que use una librería nuestra (componente JAR).

Lo primero es añadir Apache Tomcat la librería **Persona.jar** creada en apartados anteriores. Para activarlo reiniciaremos Apache Tomcat y la librería aparecerá disponible.




Luego crearemos el proyecto **AT03Persona** y copiaremos el siguiente código JSP en el archivo index.jsp.

**index.jsp**

```
<%@page import="com.dam.persona.Domicilio"%>
<%@page import="com.dam.persona.Persona"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
        <style>
            table {
                border-collapse: collapse;
            }
            table, td {
                border: 1px solid black;
            }
            .cabecera {
                background-color: orange;
                font-weight: bold;
            }

        </style>
    </head>
```

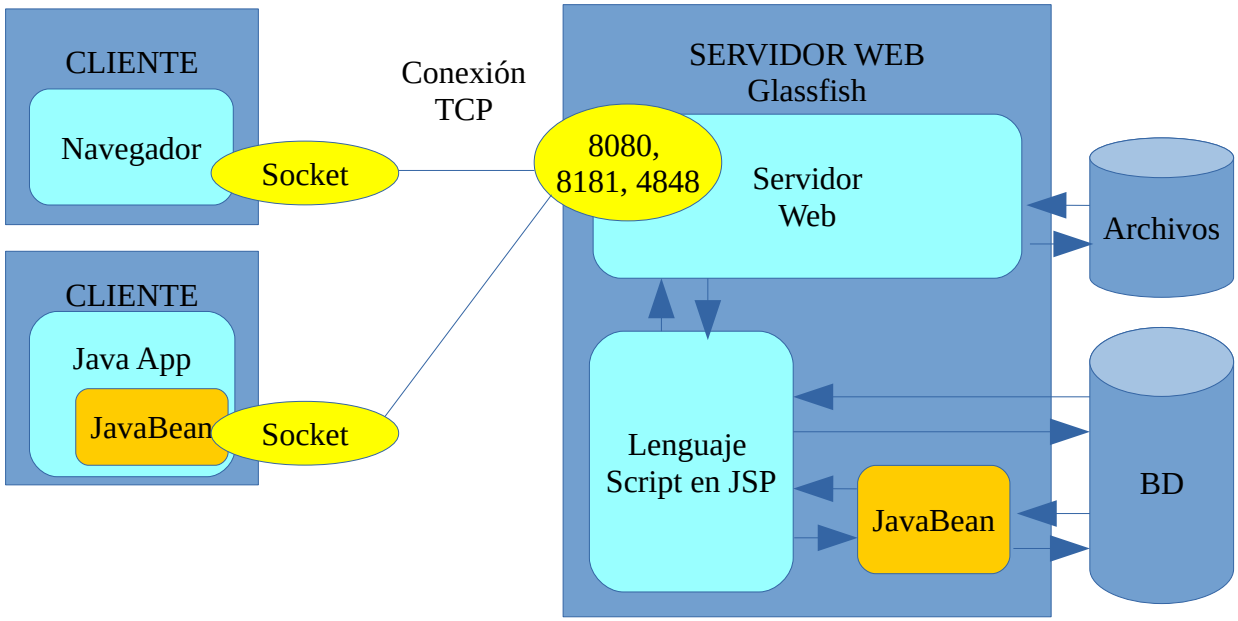
```
    <body>
        <h1>Persona</h1>
        <p><%
            Persona persona = new Persona(21444555,"Sergio", "Fernández", "61277788",
                     new Domicilio("C/Jaume de Scals, 35","03100","Xixona","Alicante"));
            out.println(persona.toString());
            %></p>
        <table>
            <tr>
                <td class="cabecera">DNI</td>
                <td><%= persona.getDni() %></td>
            </tr>
            <tr>
                <td class="cabecera">NOMBRE</td>
                <td><%= persona.getNombre() %></td>
            </tr>
            <tr>
                <td class="cabecera">APELLIDOS</td>
                <td><%= persona.getApellidos() %></td>
            </tr>
            <tr>
                <td class="cabecera">TELÉFONO</td>
                <td><%= persona.getTelefono() %></td>
            </tr>
            <tr>
                <td class="cabecera">DIRECCIÓN</td>
                <td><%= persona.getDomicilio().getDireccion() %></td>
            </tr>
            <tr>
                <td class="cabecera">CÓDIGO POSTAL</td>
                <td><%= persona.getDomicilio().getCpostal() %></td>
            </tr>
            <tr>
                <td class="cabecera">POBLACIÓN</td>
                <td><%= persona.getDomicilio().getPoblacion() %></td>
            </tr>
            <tr>
                <td class="cabecera">PROVINCIA</td>
                <td><%= persona.getDomicilio().getProvincia() %></td>
            </tr>
        </table>
    </body>
</html>
```

Como se aprecia en el ejemplo, usamos la clase **Persona** y **Domicilio** como en un proyecto "Java Application".

# 8.2 NetBeans con Glassfish

## *Instalación de Glassfish*

El servidor de glassfish tiene el mismo esquema de Apache Tomcat pero cambiando los puertos:




Para instalar Glassfish, basta con descargar el software y descomprimirlo en una carpeta del servidor.

| ***Glassfish – Web Oficial*** |
|---|
| https://javaee.github.io/glassfish/<br>https://javaee.github.io/glassfish/download |

## *Configuración Glassfish – Carpetas*

Glassfish puede tener varias instancias del servidor que se organizan en dominios. El dominio por defecto es **domain1**.

Glassfish tiene varias carpetas importantes:

| | |
|---|---|
| **glassfish5\glassfish\bin** | Utilidades |
| **glassfish5\glassfish\domains\domain1\docroot** | Alojamiento de las páginas HTML y JSP |
| **glassfish5\glassfish\domains\domain1\config** | Archivos de configuración del servidor |
| **tomcat\lib** | Archivos jar cargados en memoria por el servidor y disponibles para los archivos JSP. Cada vez que se incluya uno nuevo, es necesario reiniciar el servidor.<br>Los drivers JDBC irán en este directorio |

## ***Configuración Glassfish – Puertos***

Los puertos que usa por defecto son: 8080, 8181 y 4848.

Estos puertos pueden cambiarse en el archivo:

**`glassfish5\glassfish\domains\domain1\config\domain.xml`**

en las etiquetas "**<network-listener**".

Nosotros cambiaremos el puerto 8080 por el 8081 para poder tener Apache Tomcat y Glasfissh funcionando simultáneamente.

Recuerda que el comando para ver los puertos ocupados en windows es:

```
C:\> netstat -a -p TCP | find "LISTENING"
```

## ***Configuración Glassfish – Usuarios***

Por defecto, se puede acceder al entorno de administración si usuario y contraseña, y para el desarrollo es suficiente.

## ***Configuración Glassfish – Arrancar y parar***

Para arrancar el servicio tenemos el archivo bat:

**`glassfish5\glassfish\bin\startserv.bat domain1`**

Para parar el servicio tenemos el archivo bat:

**`glassfish5\glassfish\bin\stopserv.bat domain1`**

## ***Configuración Glassfish – Añadir componentes JAR***

Todos los archivos JAR que vayan a utilizar nuestra páginas web en JSP hay que copiarlos a la carpeta:

**`glassfish5\glassfish\lib`**

Y luego reiniciar el servicio. Netbeans nos muestra las librerías añadidas a Glashfish pero funcionan al desplegar las aplicaciones.

## ***Configuración Glassfish – URL***

Tendremos varias URL predefinidas para acceder a Glassfish:

http://localhost:8081

URL Pública que navega sobre el contenido de la carpeta glassfish5\glassfish\domains\domain1\docroot

http://localhost:4848

URL Administración que navega sobre el Manager GUI

## ***Configuración Glassfish – Prueba***

Crearemos la carpeta donde alojaremos nuestros archivos HTML, CSS, JS y JSP.

**glassfish5\glassfish\domains\domain1\docroot\ad**

| **Prueba del servidor Glashfish** |
| --- |
| Arranca el servicio de Glassfish.<br><br>Prueba a copiar **ejemplo_table.jsp** y **ejemplo_select.jsp** en la carpeta<br><br>"glassfish5\glassfish\domains\domain1\docroot\ad"<br><br>y acceder a ella desde el navegador. |

Para que funcionen el JSP de **ejemplo_table.jsp** y **ejemplo_select.jsp** correctamente en **GlassFish**, debemos modificar la llamada al constructor de datos:

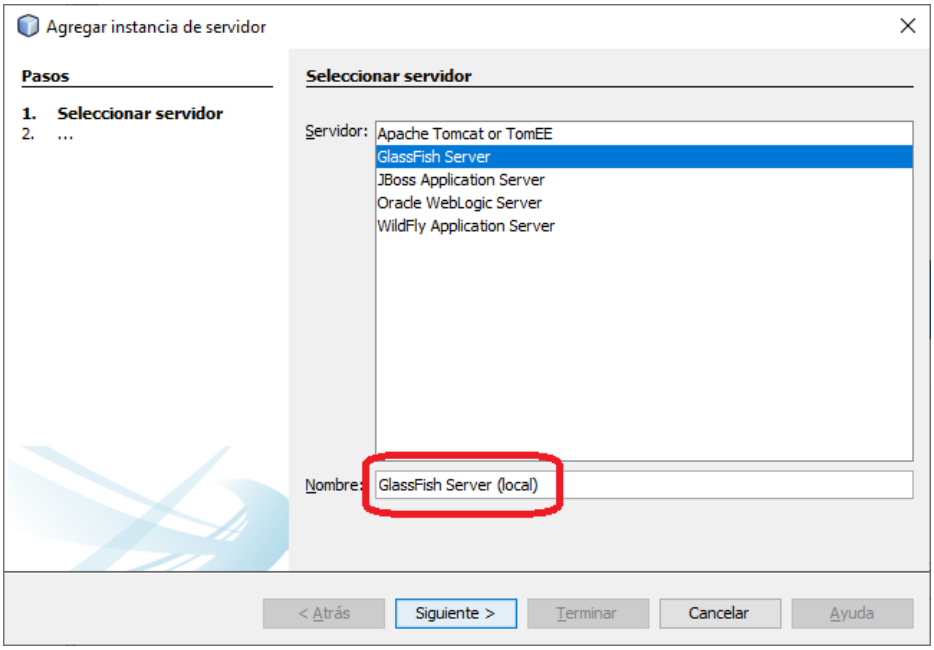
```
ArrayList<String> datos = new ArrayList<String>();
```

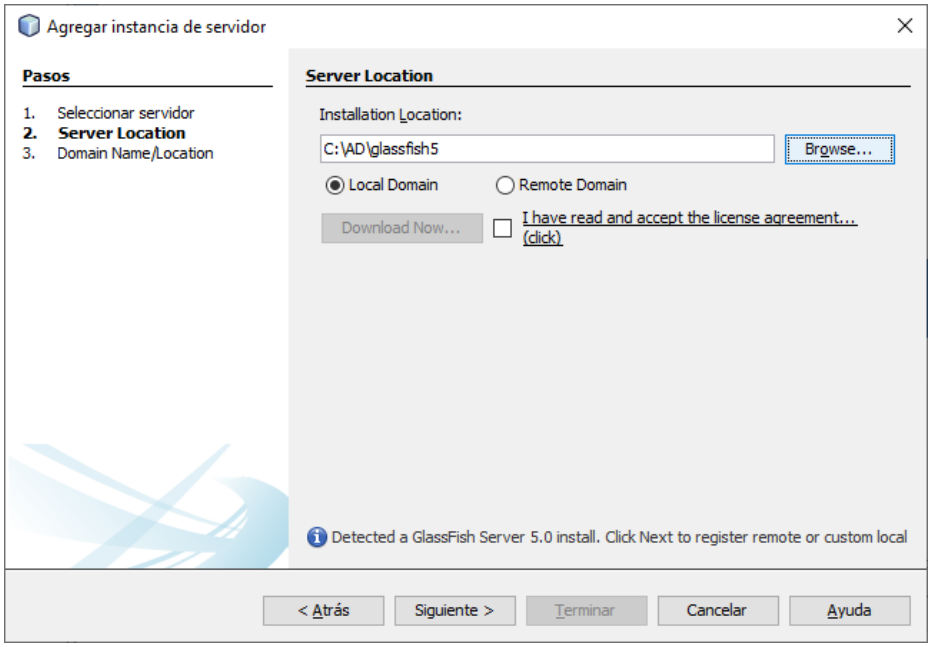
Además, el **ejemplo_select.jsp** necesita de la librería jar del driver JDBC de mysql, por lo que previamente habrá que copiarla a la carpeta de librerías de glassfish y posteriormente reiniciar el servicio:

**glassfish5\glassfish\lib\mysql-connector-java-5.1.48-bin**

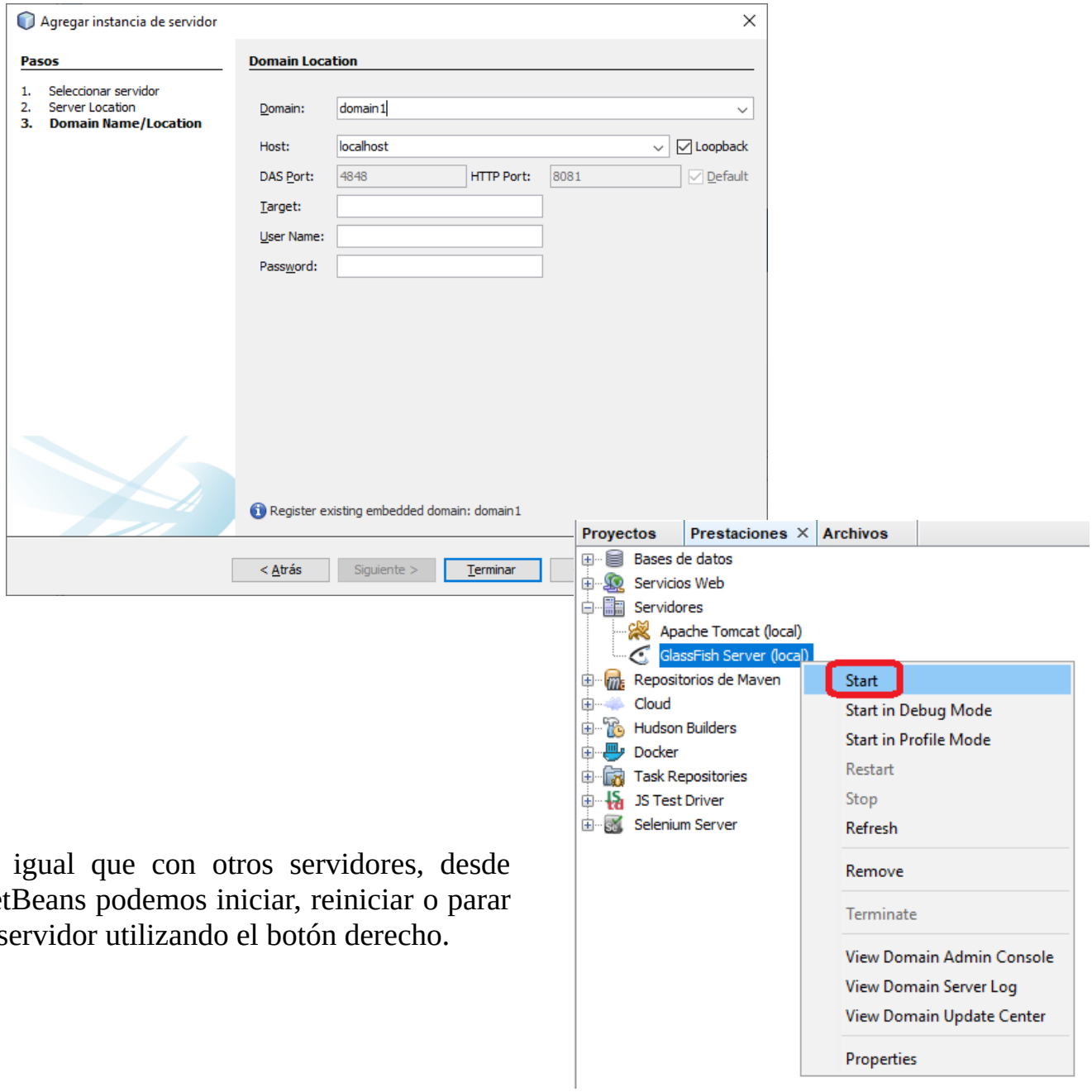
## ***Añadir el servidor Glassfish a Netbeans***

En la pestaña de Prestaciones añadiremos nuestro servidor de Glassfish:



En la última pantalla indicaremos el **domain** que vamos a añadir a Netbeans:



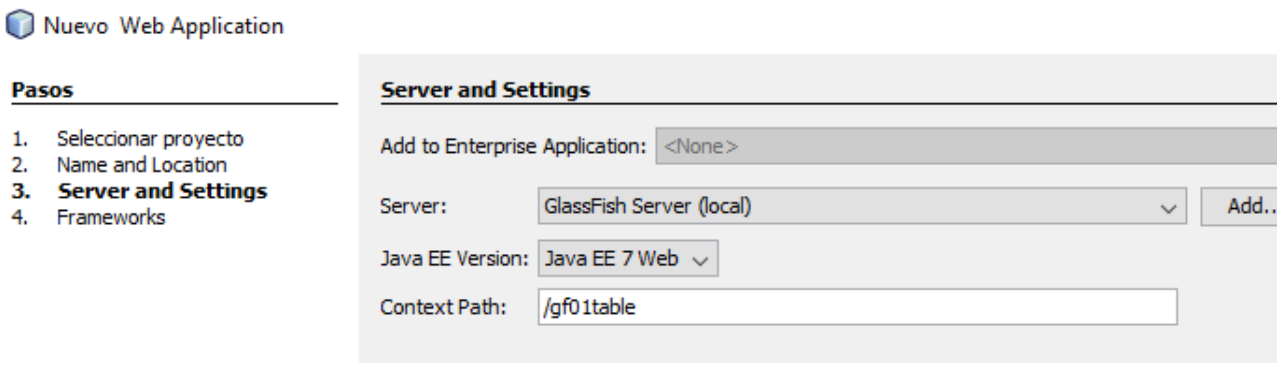

Al igual que con otros servidores, desde NetBeans podemos iniciar, reiniciar o parar el servidor utilizando el botón derecho.

## ***Crear proyecto de Java Web => Web Application***

Vamos a crear el proyecto **GF01Table** que ejecute el código de **ejemplo_table.jsp**.

Crearemos un proyecto de tipo "Java Web => Web Application" poniendo como path raíz del App Web (Context Path Root) **gf01table** y como servidor nuestro **GlassFish Server (local).**

Ahora solo queda copiar el contenido de nuestro **ejemplo_table.jsp** en el archivo **index.jsp** que ha generado NetBeans.




Cuando ejecutemos el proyecto, si GlassFish Server está parado, se iniciará (Start). Hay que tener en cuenta que al cerrar NetBeans se parará (Stop).

***Crear proyecto de Java Web => Web Application***

Vamos a crear el proyecto **GF02Select** que ejecute el código de **ejemplo_select.jsp**.

Realizaremos el mismo proceso que en el anterior proyecto, pero poniendo como path raíz del App Web (Context Path Root) **gf02select** y copiando el contenido de nuestro **ejemplo_select.jsp** en el archivo **index.jsp.**

No es necesario copiar la librería jar del driver de mysql a la aplicación para que funcione porque ya está en **glassfish5\glassfish\lib** del GlassFish Server, aunque si desplegamos las librerías incluidas en el servidor no la podremos ver.

| **Nota** |
| --- |
| A la hora de escribir el código, no se desplegarán los métodos. Si queremos tener la ayuda, incluiremos en el proyecto el jar del **JDBC de MySql** en nuestra carpeta **lib** del proyecto. |

***Crear proyecto de Java Web => Web Application***

Vamos a crear el proyecto **AT03Persona** que use una librería nuestra (componente JAR).

Lo primero es añadir GlassFish la librería **Persona.jar** creada en apartados anteriores. Para activarlo reiniciaremos GlassFish Server y aunque la librería no aparecerá como sí lo hacía en Apache Tomcat, funcionará.

| **Nota** |
| --- |
| A la hora de escribir el código, no se desplegarán los métodos de Persona. Si queremos tener la ayuda, incluiremos en el proyecto el jar de **Persona** en nuestra carpeta **lib** del proyecto. |

## 8.3 Compilación y despliegue de Aplicaciones Web

Los archivos empaquetados obtenidos en las compilaciones de nuestro proyectos Java crean archivos JAR.

Un archivo **JAR (Java ARchive)** es un tipo de archivo que permite ejecutar aplicaciones y herramientas escritas en el lenguaje Java. Las siglas están deliberadamente escogidas para que coincidan con la palabra inglesa "jar" (tarro).

Los JAR están comprimidos con el formato ZIP y cambiada su extensión a .jar.

En el caso de aplicaciones Java Web el empaquetado obtenido es WAR.

Un archivo **WAR (de Web Application Archive - Archivo de aplicación web)** es un archivo JAR utilizado para distribuir una colección de JavaServer Pages, servlets, clases Java, archivos XML, bibliotecas de tags y páginas web estáticas (HTML y archivos relacionados) que juntos constituyen una aplicación web.

Los archivos WAR se ejecutan en Servidores Web, como ya hemos visto con Apache Tomcat o Glassfish.

Para instalar un archivo WAR en un Servidor Web se realiza un **despliegue** de la aplicación web. Para ello, lo servidores web ofrecen varias formas de hacerlo:

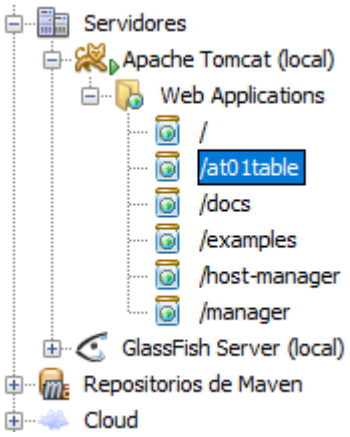
- Desde entornos IDE, como Netbeans.
- Desde una web de administración
- Desde el explorador de archivos copiando el archivo WAR en una carpeta concreta.

En inglés, **Deploy** y **Undeploy** serán las acciones para instalar y desinstalar la aplicación web.

# APACHE TOMCAT

Trabajaremos con el proyecto **AT01Table**

## *Despliegue automático desde NetBeans*




Cuando ejecutamos un Java Web en NetBeans, la aplicación se despliega automáticamente en el Servidor asociado.

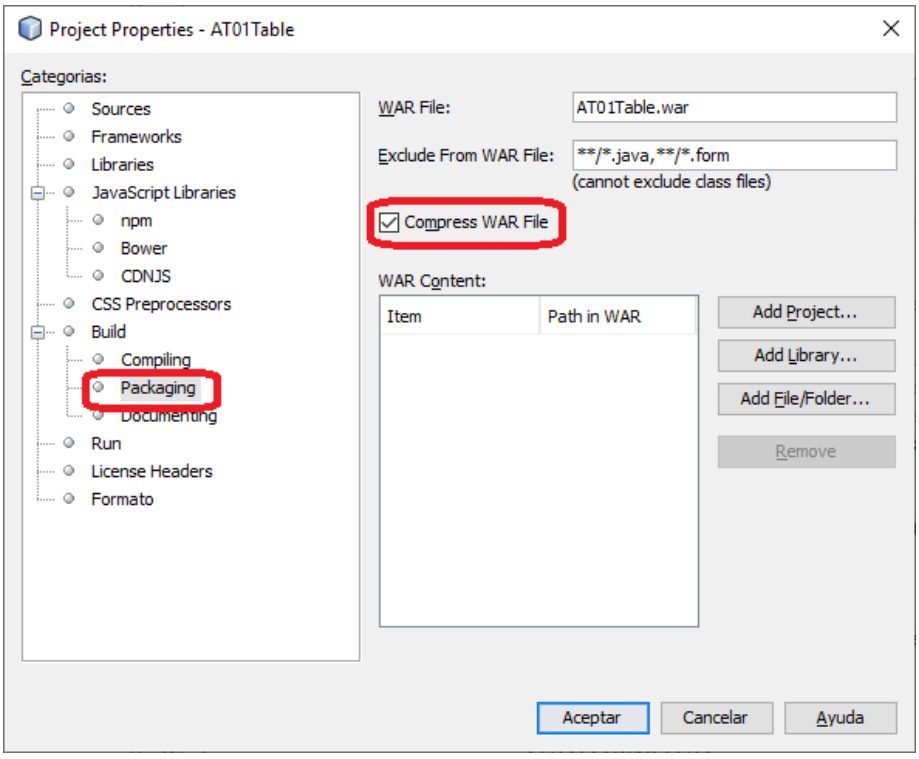
Una vez ejecutada y desplegada, podemos ver su instalación en la pestaña de "Prestaciones => Servidores".

Con botón derecho sobre la aplicación podemos desinstalarla con **Undeploy**.

## *Obtener empaquetado (Archivo WAR)*

Para distribuir nuestra aplicación, necesitaremos el archivo WAR.

Activaremos la generación del archivo desde Propiedades del proyecto en:




Para obtener el archivo empaquetado tendremos que general el proyecto con el icono del martillo o F11.

El fichero **AT1Table.war** estará en la carpeta **dist** de nuestro proyecto.

## ***Despliegue manual de WAR en Apache Tomcat***

Para desplegar manualmente nuestro WAR en un servidor Apache Tomcat lo copiaremos en la carpeta **webapps**, y automáticamente se desplegará y creará la carpeta de la aplicación web con los JSP que hayamos creado desde NetBeans.

Podemos probar su funcionamiento navegando sobre la carpeta del proyecto:

`http://localhost:8080/AT01Table`

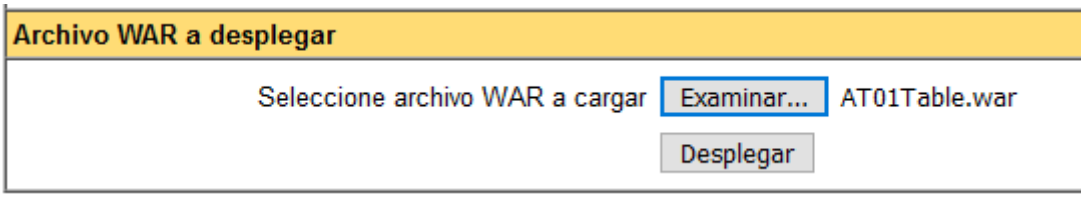
## ***Despliegue de WAR desde Manager APP de Apache Tomcat***

Para desplegar nuestro WAR utilizando el Manager App accederemos a:

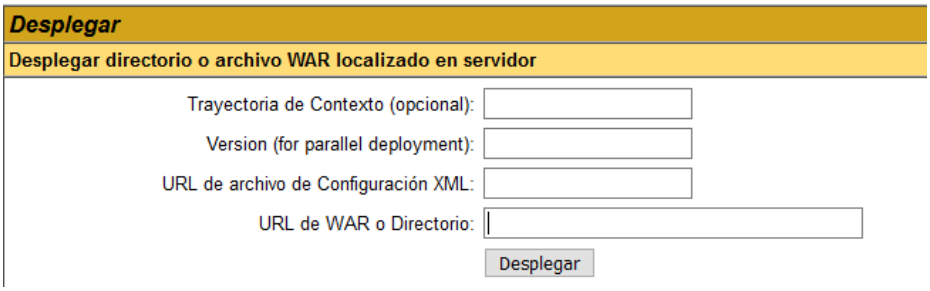
`http://localhost:8080/manager/html`

y nos pedirá el usuario de **tomcat** que creamos en la instalación con la contraseña **1234**.

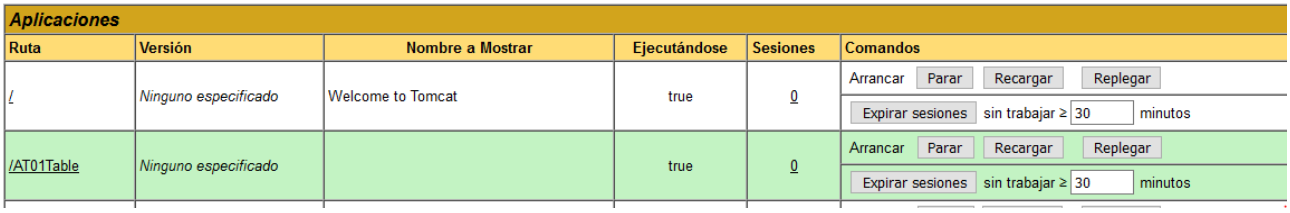
Desde esta consola podremos **instalar** (Desplegar/Deploy) aplicaciones **WAR**:




También podríamos subirlo por ftp/sftp al servidor y desplegarlo, pudiendo cambiar su trayectoria de contexto (Context Path) por defecto:




y **desinstalar** (Replegar/Undeploy):

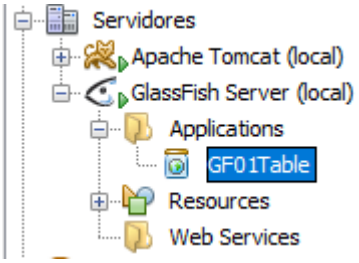


# GLASSFISH

Trabajaremos con el proyecto **GF01Table**, siendo el proceso muy parecido al de Apache Tomcat.

## *Despliegue automático desde NetBeans*

Cuando ejecutamos un Java Web en NetBeans, la aplicación se despliega automáticamente en el Servidor asociado.




Una vez ejecutada y desplegada, podemos ver su instalación en la pestaña de "Prestaciones => Servidores".

Con botón derecho sobre la aplicación podemos desinstalarla con **Undeploy**.

## *Obtener empaquetado (Archivo WAR)*

Ya explicado en Apache Tomcat

## *Despliegue manual de WAR en Apache Tomcat*

Para desplegar manualmente nuestro WAR en un servidor GlassFish lo copiaremos en la carpeta:

```
glassfish5\glassfish\domains\domain1\autodeploy
```

y automáticamente se desplegará y creará la carpeta de la aplicación web con los JSP que hayamos creado desde NetBeans.

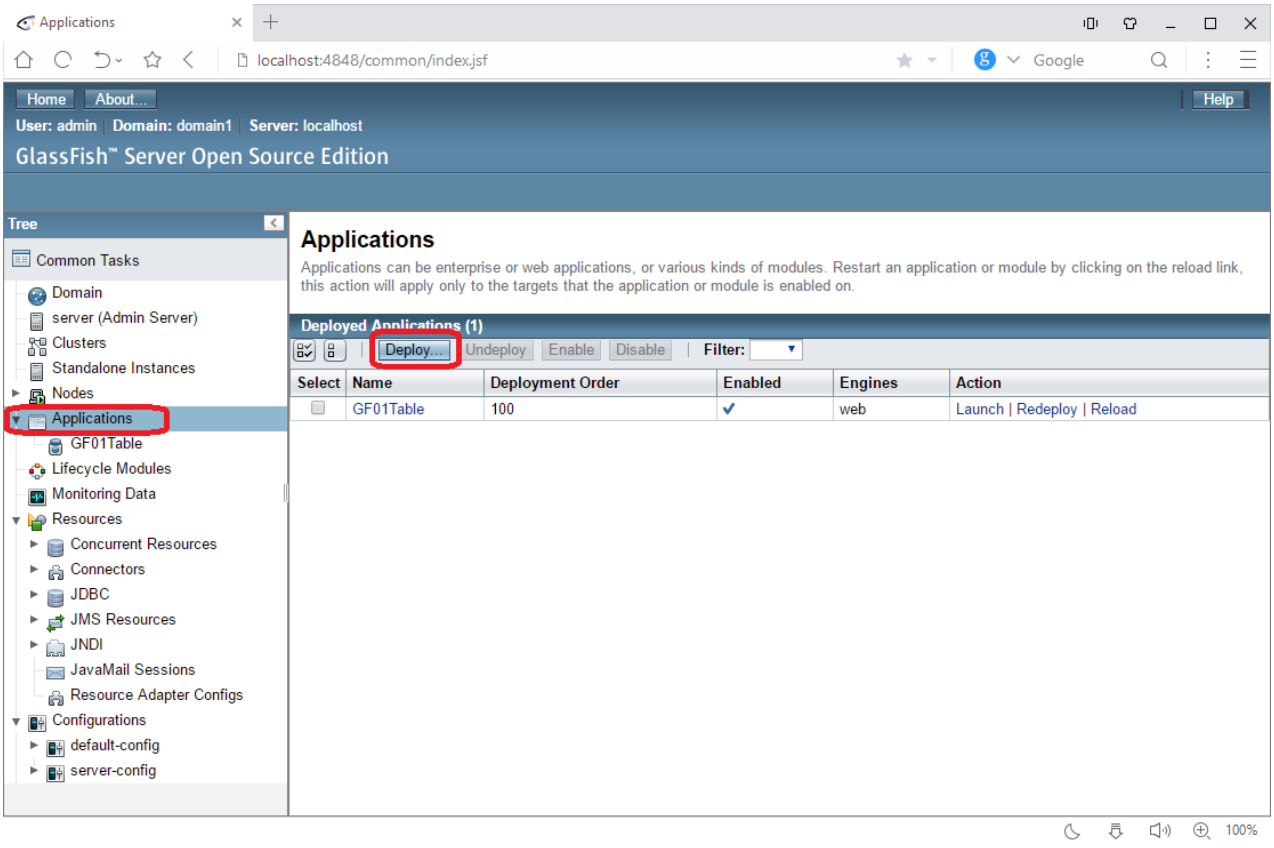
Podemos probar su funcionamiento navegando sobre la carpeta del proyecto:

http://localhost:8081/gf01table/

## ***Despliegue de WAR desde Manager APP de GlassFish***

Para desplegar nuestro WAR utilizando el Manager App accederemos a:

http://localhost:4848




Desde la pestaña de Applications podremos desplegar (Deploy) aplicaciones a partir del archivo WAR, así como desinstalar (Undeploy) marcando la aplicación: